Notebook

November 7, 2016

Test Name	Score	Possible	Percent Tests Passed
q10	1	1	100
q20	1	1	100
q200	1	1	100
q210	1	1	100
q220	1	1	100
q230	1	1	100
q30	1	1	100
q300	1	1	100
q301	0	1	0
q40	1	1	100
q41	1	1	100
q42	1	1	100
q43	1	1	100
q500	3	3	100
q5000	0	1	0
q501	0	1	0
q502	0	1	0
q510	3	3	100
q5100	3	3	100
q511	3	3	100
q512	3	3	100
q520	1	1	100
q521	1	1	100
q530	1	1	100
q60	1	1	100
q61	1	1	100

DETECTED **Question 1.0.** In the next cell, assign the name `new_year` to the larger number among the following two numbers: $|2^5 - 2^{11}|$ and $5 \times 13 \times 31$. Try to use just one statement (one line of code).

```
In [2]: new_year = max(abs(2**5 - 2**11), 5*13*31) #SOLUTION
        new_year

Out[2]: 2016
```

DETECTED **Question 2.0.** Replace each `...` with a single-word string literal below so that the final expression prints its punchline.

```
In [5]: why_was = "Because"
        six = "seven" # SOLUTION
        afraid_of = "eight"
        seven = "nine" # SOLUTION
        print(why_was, six, afraid_of, seven)

Because seven eight nine
```

DETECTED **Question 2.0.0** Assign strings to the names `you` and `this` so that the final expression evaluates to a 10-letter English word with three double letters in a row. *Hint*: Ask your neighbors (or Google) if you can't think of a word that has three double letters in a row.

```
In [11]: you = 'keep' # SOLUTION
         this = 'book' # SOLUTION
         a = 'beeper'
         the = a.replace('p', you)
         the.replace('bee', this)

Out[11]: 'bookkeeper'
```

DETECTED **Question 2.1.0.** Assign `good` to a string that makes the final expression below evaluate to the integer 99.

```
In [17]: good = "1" # SOLUTION
         lol = "L" + str(0) + "L"
         int(lol.replace("L", good)) - 2

Out[17]: 99
```

DETECTED

0.1 2.2. String Arguments

String values, like numbers, can be arguments to functions and can be returned by functions. The function `len` takes a single string as its argument and returns the number of characters in the string: its **len**gth.

Question 2.2.0. Use `len` to find out the length of the very long string in the next cell. (It's the first sentence of the English translation of the French Declaration of the Rights of Man.) The length of a string is the total number of characters in it, including things like spaces and punctuation. Assign `sentence_length` to that number.

```
In [19]: a_very_long_sentence = "The representatives of the French people, organize
         sentence_length = len(a_very_long_sentence) #SOLUTION
         sentence_length

Out[19]: 896
```

DETECTED **Question 2.3.0.** Using `type`, assign `sentence_type` to the type of `a_very_long_sentence`.

```
In [20]: sentence_type = type(a_very_long_sentence) #SOLUTION
         sentence_type

Out[20]: str
```

DETECTED **Question 3.0.** `math` also provides the name `e` for the base of the natural logarithm, which is roughly 2.71. Compute $e^{\pi} - \pi$, giving it the name `near_twenty`.

```
In [22]: near_twenty = math.e ** math.pi - math.pi # SOLUTION
         near_twenty

Out[22]: 19.99909997918947
```

DETECTED

0.2 3.0. Importing functions

Modules can provide other named things, including functions. For example, `math` provides the name `sin` for the sine function. Having imported `math` already, we can write `math.sin(3)` to compute the sine of 3. (Note that this sine function considers its argument to be in radians, not degrees. 180 degrees are equivalent to π radians.)

Question 3.0.0. Compute the sine of $\frac{\pi}{4}$ using `sin` and `pi` from the `math` module. Give the result the name `sine_of_pi_over_four`.

(Source: Wolfram MathWorld)

```
In [23]: sine_of_pi_over_four = math.sin(math.pi / 4) #SOLUTION
         sine_of_pi_over_four

Out[23]: 0.7071067811865475
```

DETECTED People have written Python functions that do very cool and complicated things, like crawling web pages for data, transforming videos, or doing machine learning with lots of data. Now that you can import things, when you want to do something with code, first check to see if someone else has done it for you. Let's see an example of a function that's used for downloading and displaying pictures.

The module `IPython.display` provides a function called `Image`. `Image` takes a single argument, a string that is the URL of the image on the web. It returns an *image* value that this Jupyter notebook understands how to display. To display an image, make it the value of the last expression in a cell, just like you'd display a number or a string.

Question 3.0.1. In the next cell, import the module `IPython.display` and use its `Image` function to display the image at this URL:

```
https://upload.wikimedia.org/wikipedia/commons/thumb/8/8c/David_-_The_Death_of_Socr
```

Give the name `art` to the output of the function call, then make the last line of the cell `art` to see the image. (It might take a few seconds to load the image. It's a painting called *The Death of Socrates* by Jacques-Louis David, depicting events from a philosophical text by Plato.)

DETECTED **Question 4.0.** Check whether 3^{10} is larger than 4^9. (Count it as larger than 4^9 if they're both equal.) Give the result of the comparison (a `True` or `False` value) the name `three_to_the_tenth_is_larger`.

```
In [26]: three_to_the_tenth_is_larger = 3**10 > 4**9 #SOLUTION
         three_to_the_tenth_is_larger
```

```
Out[26]: False
```

DETECTED **Question 4.1.** Check whether $2 + 2$ equals 4. Give the result of the comparison the name `two_plus_two_is_four`.

```
In [27]: two_plus_two_is_four = 2 + 2 == 4 #SOLUTION
         two_plus_two_is_four

Out[27]: True
```

DETECTED **Question 4.2.** The == operator works on many values, including strings. The next cell has two quotes from a famous computer scientist, Donald Knuth. Use it to check whether the two quotes are the same. Assign the result the name `quotes_are_the_same`.

```
In [28]: a_quote = "Let us change our traditional attitude to the construction of p
         another_quote = "let us change oUr Traditional attitude to the Constructio
         quotes_are_the_same = a_quote == another_quote #SOLUTION
         quotes_are_the_same

Out[28]: False
```

DETECTED **Question 4.3.** The passages look the same, and in fact they *are* the same, except that the capitalization of `another_quote` has been mangled. Python considers two strings with the same letters and different capitalization to be different. Use the string method `lower` to check whether the two strings are the same *after they've both been lowercased*. (This is a way to check whether two strings are the same without counting capitalization differences.) Assign the result the name `quotes_have_the_same_letters`.

```
In [29]: quotes_have_the_same_letters = a_quote.lower() == another_quote.lower() #S
         quotes_have_the_same_letters

Out[29]: True
```

DETECTED **Question 5.0.0.** Make a list of lists containing `list_of_strings` and `another_list_of_booleans`. The output should look like:

```
[['Hello', 'world', '!'], [True, False]]
```

```
In [ ]: list_of_lists = [list_of_strings, another_list_of_booleans] #SOLUTION
        list_of_lists
```

DETECTED **Question 5.1.0.** Make an array containing the numbers 1, 2, and 3, in that order. Name it `small_numbers`.

```
In [ ]: small_numbers = np.array([1, 2, 3]) #SOLUTION
        small_numbers
```

DETECTED **Question 5.1.1.** Make an array containing the numbers 0, 1, -1, π, and e, in that order. Name it `interesting_numbers`. *Hint:* How did you get the values π and e earlier? You can refer to them in exactly the same way here.

```
In [ ]: interesting_numbers = np.array([0, 1, -1, math.pi, math.e]) #SOLUTION
        interesting_numbers
```

DETECTED **Question 5.1.2.** Make an array containing the five strings `"Hello"`, `","`, `" "` (that's just a single space inside quotes), `"world"`, and `"!"`. Name it `hello_world_components`.

Note: If you print `hello_world_components`, you'll notice some extra information in addition to its contents: `dtype='<U5'`. That's just NumPy's extremely cryptic way of saying that the things in the array are strings.

```
In [ ]: hello_world_components = np.array(["Hello", ",", " ", "world", "!"]) #SOLU
        hello_world_components
```

DETECTED **Question 5.1.0.0.** NOAA (the US National Oceanic and Atmospheric Administration) operates weather stations that measure surface temperatures at different sites around the United States. The hourly readings are publicly available. Suppose we download all the hourly data from the Oakland, California site for the month of December 2015, and we find that the data don't include the timestamps of the readings (the time at which each one was taken). We'll assume the first reading was taken at the first instant of December 2015 (midnight on December 1st) and each subsequent reading was taken exactly 1 hour after the last.

Create an array of the *time, in seconds, since the start of the month* at which each hourly reading was taken. Name it `collection_times`.

Hint: There were 31 days in December, which is equivalent to 31×24 hours or $31 \times 24 \times 60 \times 60$ seconds. So your array should have 31×24 elements in it.

Hint 2: The `len` function works on arrays, too. Check the length of `collection_times` and make sure it has 31×24 elements.

```
In [ ]: collection_times = np.arange(0, 31*24*60*60, 60*60) #SOLUTION
        collection_times
```

DETECTED **Question 5.2.0.** What is the index of the first element of `interesting_numbers`? Set `index_of_first_element` to that index (a number).

```
In [ ]: index_of_first_element = 0 #SOLUTION
```

DETECTED **Question 5.2.1.** Set `last_interesting_number` to the last element of `interesting_numbers`, using `item`.

```
In [ ]: last_interesting_number = interesting_numbers.item(4) #SOLUTION
        last_interesting_number
```

DETECTED Here is one simple question we might ask about world population: How big is it *in orders of magnitude*? The logarithm function is one way of measuring how big a number is. The logarithm (base 10) of a number increases by 1 every time we multiply the number by 10. It's like a measure of how many digits the number has.

NumPy provides a function called `log10` that takes the logarithm of each element of an array. It takes a single array of numbers as its argument and returns an array of the same length, where the first element of the result is the logarithm of the first element of the argument, and so on.

Question 5.3.0. Set `log_world_population` to an array whose first element is the logarithm of the population in 1950, whose second element is the logarithm of the population in 1951, and so on. This would take quite awhile to do by hand. Your code should be very short.

```
In [ ]: log_world_population = np.log10(world_population) #SOLUTION
        log_world_population
```

DETECTED **Question 6.0.** The Greek philosopher Plato wrote many works called *dialogues*. His thinking evolved over the years in which he wrote, and Plato scholars divide his dialogues into Early, Middle, and Late periods.

The following cell creates two arrays of strings. `early_dialogues` is an array of the titles of the Early dialogues, and `middle_dialogues` is an array of the titles of the Middle dialogues. (The data are from the Internet Sacred Text Archive.) Set `more_early_dialogues` to a boolean value that is `True` if there are more Early dialogues than Middle dialogues, and `False` otherwise. (If there is a tie, set it to `False`.)

Hint: Remember the `len` function? It works on arrays, too.

```
In [ ]: early_dialogues = np.array(["Apology", "Crito", "Charmides", "Laches", "Lys
        middle_dialogues = np.array(["Gorgias", "Protagoras", "Meno", "Euthydemus",
        more_early_dialogues = len(early_dialogues) > len(middle_dialogues) #SOLUTI
        more_early_dialogues
```

DETECTED **Question 6.1.** The US Census Bureau estimates (to which, recall, we gave the name `world_population`) are not the only estimates of world population. The United Nations Department of Economic and Social Affairs also produces estimates, and they are slightly different. `un_world_population` in the next cell contains all the estimates since 1950.

Next time you'll learn how to write code to compare all the estimates. But the first thing we should check when investigating the two datasets is whether we have data from the same number of years in both! Set `same_number_of_years` to `True` if we do, and `False` otherwise.

```
In [ ]: un_world_population = np.array([2525149312, 2571867515, 2617940399, 266402
        same_number_of_years = len(un_world_population) == len(world_population) #
        same_number_of_years
```

```
In [ ]: # Run this cell to submit your work *after* you have passed all of the test
        # It's ok to run this cell multiple times. Only your final submission will

        #!TZ=America/Los_Angeles ipython nbconvert --output=".lab02_$(date +%m%d_%
```

```
In [ ]: import gsExport
        gsExport.generateSubmission()
```